# NORMAS DEL Lenguaje C (OBLIGACIÓN USARLAS EN ESTA ASIGNATURA)

Algunas de estas normas y recomendaciones pueden variar al emplear otro lenguaje de programación, pero el objetivo será siempre el mismo: conseguir un código satisfactorio, es decir, además de las características fundamentales, debe ser eficaz, legible y sencillo de modificar/ampliar.

## Nomenclatura de los identificadores (nombres):

1) Los identificadores de constantes, variables, registros y funciones deben ser descriptivos en relación a su existencia o a lo que almacena.
2) Los identificadores de variables, constantes y estructuras suelen ser sustantivos mientras que los identificadores de funciones son verbos.
3) No utilizar identificadores muy largos, ni tampoco tan cortos porque se pierda su significado. En cualquier caso, se dará preferencia a la claridad.
4) C es un lenguaje que distingue entre mayúsculas y minúsculas. Al establecer el identificador de constantes, variables, registros o funciones, se recomienda considerar lo siguiente:
   - Los identificadores de las variables, registros y funciones se escribirán en minúsculas, pero si el nombre consta de varias palabras, la inicial de la segunda y subsiguientes se escribirán en mayúsculas.
   - Si la variable es de tipo puntero su nombre empezará por "ptr".
   - Los identificadores de las constantes siempre deben ser mayúsculas. Si el nombre consta de varias palabras se escribirán separadas por un guion bajo.

**Ejemplos:**

```
#define MAX_FILAS 20
int     dia, numAlumnos;
float   notaMedia;
int     *ptrVar;    //Puntero que apunta a una variable  tipo entera
int     guardarPuntoCartesiano (void);
struct  punto {
    float x;
    float y;
}
```

**5)** Las variables deben declararse al inicio de la función dueña, respetando la nomenclatura indicada anteriormente. **Nunca se deben usar variables globales.**

6) Las variables para el control de iteraciones (de tipo entero) tendrán nombres cortos como i, j ó k
7) Se debe evitar el uso de una misma variable para almacenar dos o más cosas distintas.
8) Las variables de un mismo tipo se pueden declarar en una misma línea de código, pero sólo se recomienda alinear aquellas variables que tienen un significado similar.
9) Siempre deben estar inicializadas. Además, es aconsejable que la inicialización se realice junto a la declaración.

**Ejemplos:**

```
int   i, j, k;        //Variables de control
int   num ;           //Almacena un número introducido por teclado.
int   contador=0;     //inicializar en la misma línea que se define
```

## Expresiones:

10) Evitar expresiones numéricas o condiciones complejas. Dividir en sub-expresiones..
11) En una estructura condicional, ***if*** llevará la condición que se cumple más comúnmente. La menos común será para el ***else***.
12) Aclarar el orden de las operaciones involucradas en una expresión con los paréntesis correspondientes.
13) Evitar el anidamiento excesivo de estructuras condicionales (if, switch) y/o iterativas (for, while, do-while). Si hay más de estas tres estructuras anidadas emplear funciones para reducir el número de ellas.
14) Un conjunto de instrucciones no trivial que se repite a lo largo del código, ha de ser reemplazado por la correspondiente función.

## Funciones:

15) Debe evitarse el diseño de funciones demasiado largas. En el caso de los subprogramas, más de dos pantallas de código se puede considerar como demasiado largo.
16) En una función todas las variables deben declararse como parámetros o como variables locales.
17) Si una función no acepta parámetros debe indicarse con la palabra reservada *void*, es decir:
    int mostrarMenu(***void***); y debe retornar un valor igual al tipo de retorno de la función.
18) Si una función no devuelve ningún valor, debe declararse como tipo *void*, es decir:
    ***void*** mostrarDatos(int a, int b); y NO debe tener la palabra reservada ***return***.
19) En una función, que tiene que devolver un resultado, se recomienda incluir siempre una única sentencia return y que ésta esté situada al final de la función.

## Arreglos y Estructuras (Registros):

20) El tamaño de un vector debe estar declarado como una constante.

**Ejemplo**:

```
#define TAM 20
Int main(void) {
    int vector[TAM];
    …
    return 0;
}
```

21) Declarar todas las estructuras inmediatamente después de las directivas #include.
22) Cada campo de una estructura debe declararse en un renglón separado.

**Ejemplo**:

```
#define TAM 20
struct tipoLibro {
    char   titulo [TAM];
    char   autor[TAM];
    int    edicion;
}
```

23) Cuando un vector se pasa como parámetro a una función, es buena práctica pasar el tamaño del vector como parámetro adicional. Con esto se consigue dar mayor generalidad a las funciones.

24) Los campos de un registro deben reflejar de forma natural las entidades que lo componen. Se recomienda por tanto no usar construcciones de tipos denominados "anónimos". Por el contrario, se recomienda declarar las componentes e identificarlas una a una.

**Ejemplo**:

```
#define TAM 20
struct tipoAutor {
    char   nombre [TAM];
    char   apellido[TAM];
}

struct tipoLibro {
    char         titulo [TAM];
    tipoAutor    autor;
    int          edicion;
}
```

## Edición:

25) No escribir líneas de más de 70 caracteres. En caso contrario, partir la línea ya sea comentario o instrucción de código.

26) Emplear dos/tres espacios como medida básica de sangrado, no emplear el carácter tabulador (#09 de la tabla ASCII).

27) Las instrucciones de un mismo bloque deben ir todas ellas con el mismo sangrado. El cuerpo de una estructura condicional o iterativa se tabulará con dos espacios a la derecha con respecto a la línea "cabecera" de la instrucción. Ejemplo:

```
int main (void){
  instrucción_1;
  instrucción_2;
  ...
  if (expresión){
    instrucción_if_1;
    instrucción_if_2;
    …….
  } //fin if
  ...
  return 0;
};
```

28) La llave, "{", que marca el inicio de un bloque de código, se debe incluir a la derecha de la sentencia que introduce el bloque. Por otro lado, la llave, "}", que marca el final del bloque, debe colocarse en la misma columna que dicha sentencia (igual tabulado) pero en una línea que no tenga código ejecutable. Además, resulta muy aconsejable marcar con comentarios el final de cada bloque (Ver ejemplo anterior).

29) Se aconseja no utilizar más de una instrucción por línea de código, aunque es permitido agrupar en una misma línea sentencias similares.

**Ejemplo**:

```
printf ("Introduzca un número ");     scanf ("%d", &n);
```

## Comentarios:

30) Todo programa debería encabezarse con los datos que permitan identificar al autor, con un breve resumen de los objetivos que ha de satisfacer y los correspondientes campos de auditoría.

**Ejemplo**:

```
/*****************************************************************
*  Programa:                                                     *
*  Objetivo:                                                     *
*  Fecha Creación:          Fecha última actualización:          *
*  Autor/Grupo:                                                  *
*  Asignatura:              Sección:                             *
*****************************************************************/
```

31) Se aconseja que las funciones tengan una pequeña descripción y las fechas de sus actualizaciones.

32) Se deben añadir explicaciones a todo lo que no es evidente pero también hay que evitar las redundancias. Es decir, no hay que repetir lo que se hace, sino explicar, de forma clara y concisa, por qué se hace.